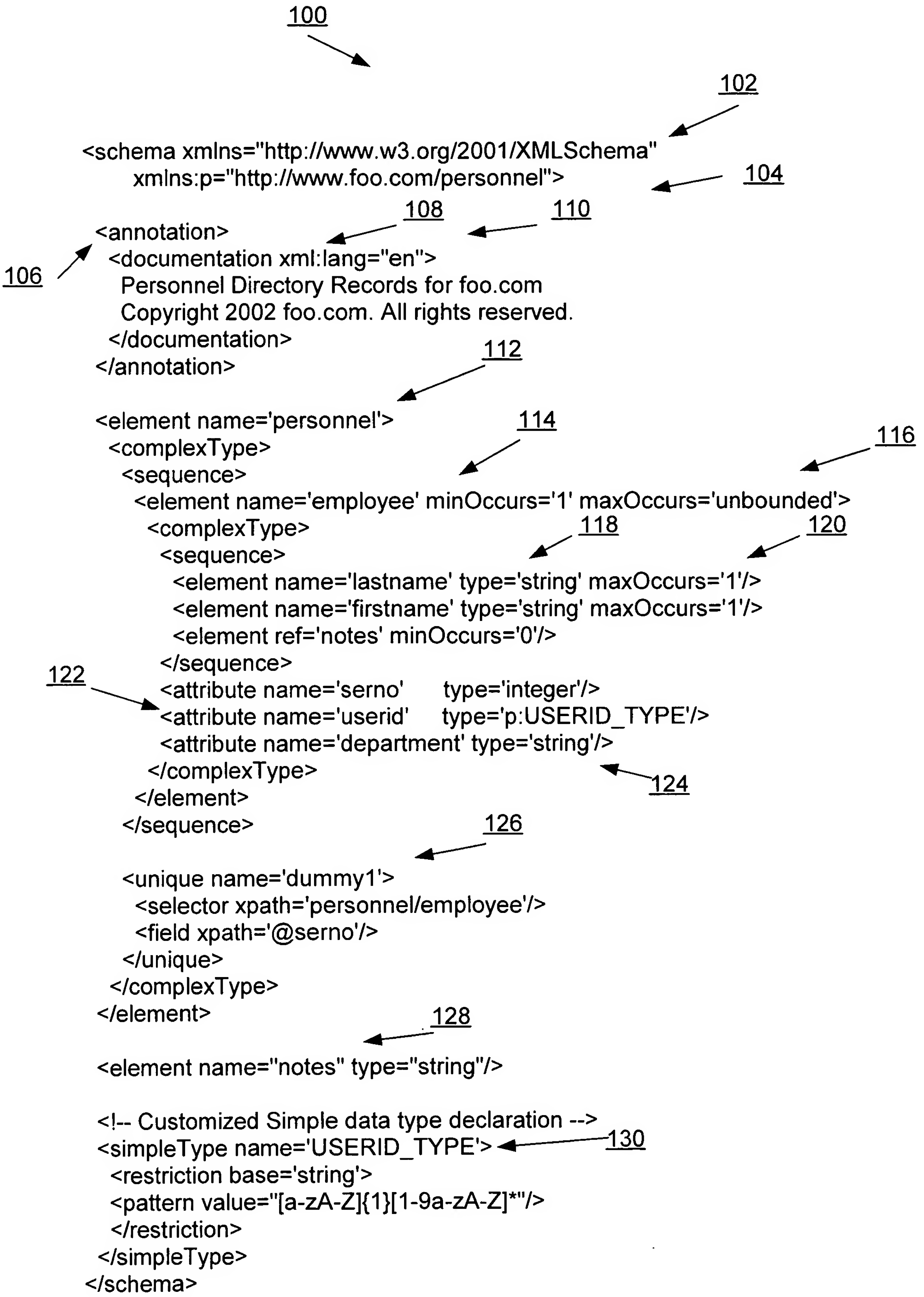


```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
      xmlns:p="http://www.foo.com/personnel">

 <annotation>
  <documentation xml:lang="en">
   Personnel Directory Records for foo.com
   Copyright 2002 foo.com. All rights reserved.
  </documentation>
 </annotation>

 <element name='personnel'>
  <complexType>
   <sequence>
    <element name='employee' minOccurs='1' maxOccurs='unbounded'>
     <complexType>
      <sequence>
       <element name='lastname' type='string' maxOccurs='1'/>
       <element name='firstname' type='string' maxOccurs='1'/>
       <element ref='notes' minOccurs='0'/>
      </sequence>
      <attribute name='serno'      type='integer'/>
      <attribute name='userid'     type='p:USERID_TYPE'/>
      <attribute name='department' type='string'/>
     </complexType>
    </element>
   </sequence>

   <unique name='dummy1'>
    <selector xpath='personnel/employee'/>
    <field xpath='@serno'/>
   </unique>
  </complexType>
 </element>

 <element name="notes" type="string"/>

 <!-- Customized Simple data type declaration -->
 <simpleType name='USERID_TYPE'>
  <restriction base='string'>
  <pattern value="[a-zA-Z]{1}[1-9a-zA-Z]*"/>
  </restriction>
 </simpleType>
</schema>
```

100
102
104
106
108
110
112
114
116
118
120
122
124
126
128
130

FIGURE 1

```
<personnel>:= PS <employee> PE

<employee>:=  <employee>

<employee>:=  <employees> <employee>

<employee>:=  ES <lastname> <firstname> <notes_01> EE

<lastname> :=  LS LE

<firstname> :=  FS FE

<notes>      :=  NS NE

<notes_01> :=  epsilon | <notes>
```

FIGURE 2

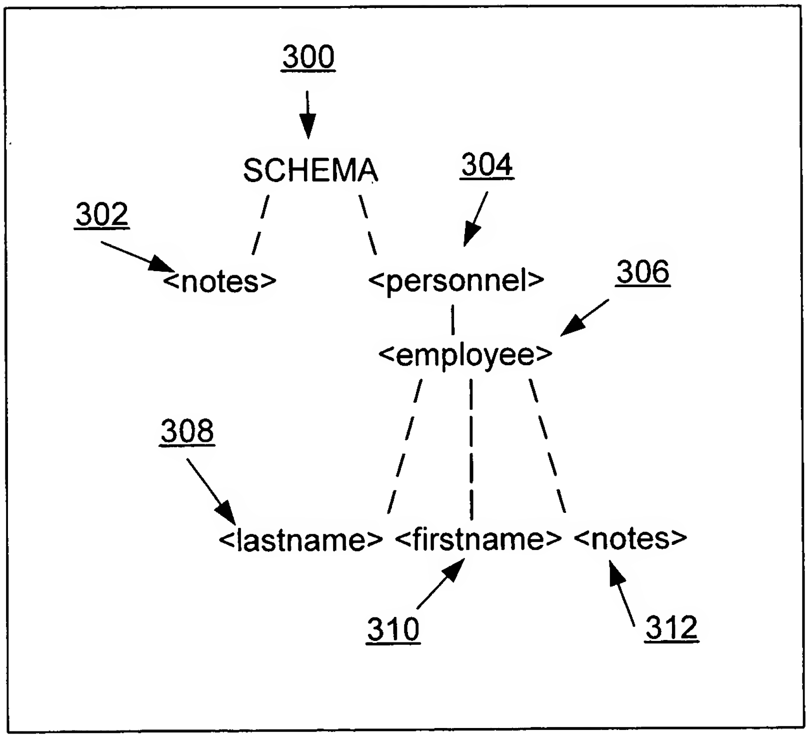
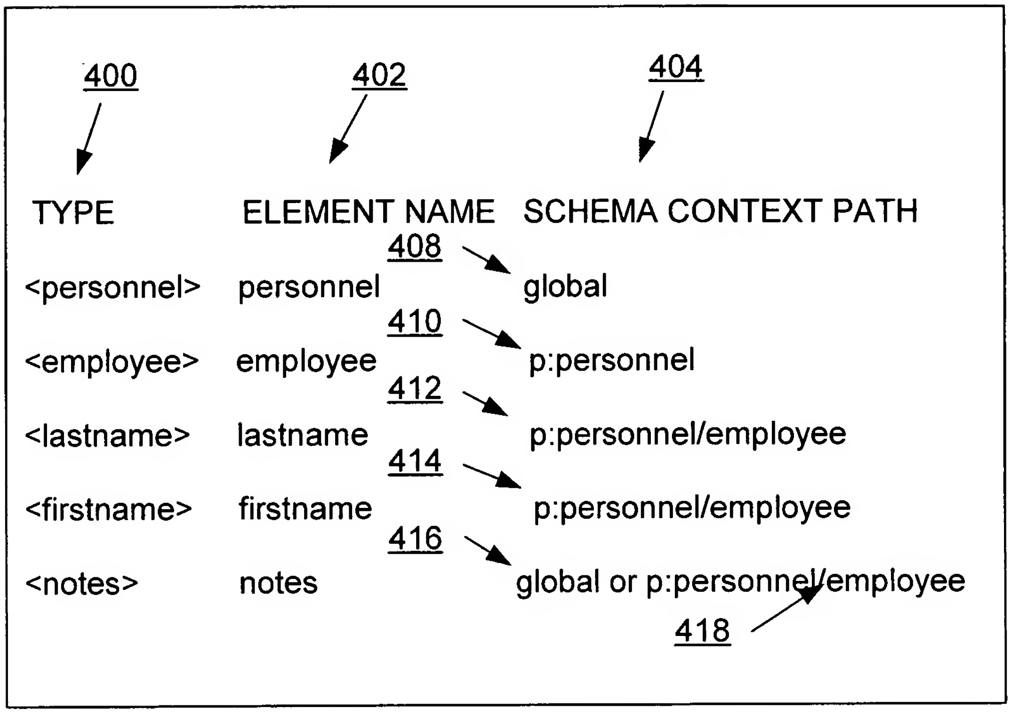


FIGURE 3



TYPE	ELEMENT NAME	SCHEMA CONTEXT PATH
<personnel>	personnel	global
<employee>	employee	p:personnel
<lastname>	lastname	p:personnel/employee
<firstname>	firstname	p:personnel/employee
<notes>	notes	global or p:personnel/employee

FIGURE 4

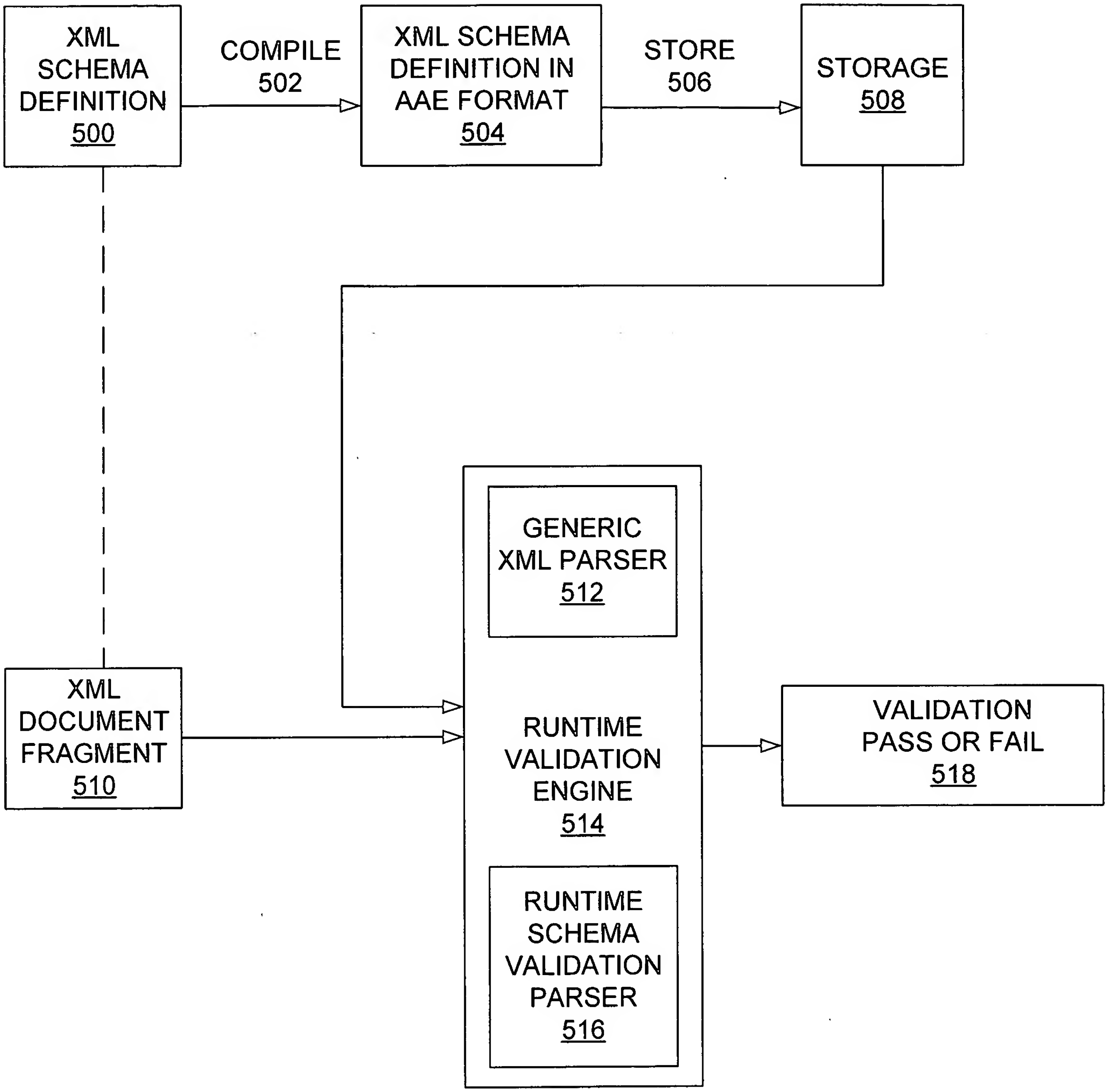

XML SCHEMA DEFINITION 500
COMPILE 502
XML SCHEMA DEFINITION IN AAE FORMAT 504
STORE 506
STORAGE 508
GENERIC XML PARSER 512
XML DOCUMENT FRAGMENT 510
RUNTIME VALIDATION ENGINE 514
VALIDATION PASS OR FAIL 518
RUNTIME SCHEMA VALIDATION PARSER 516

FIGURE 5

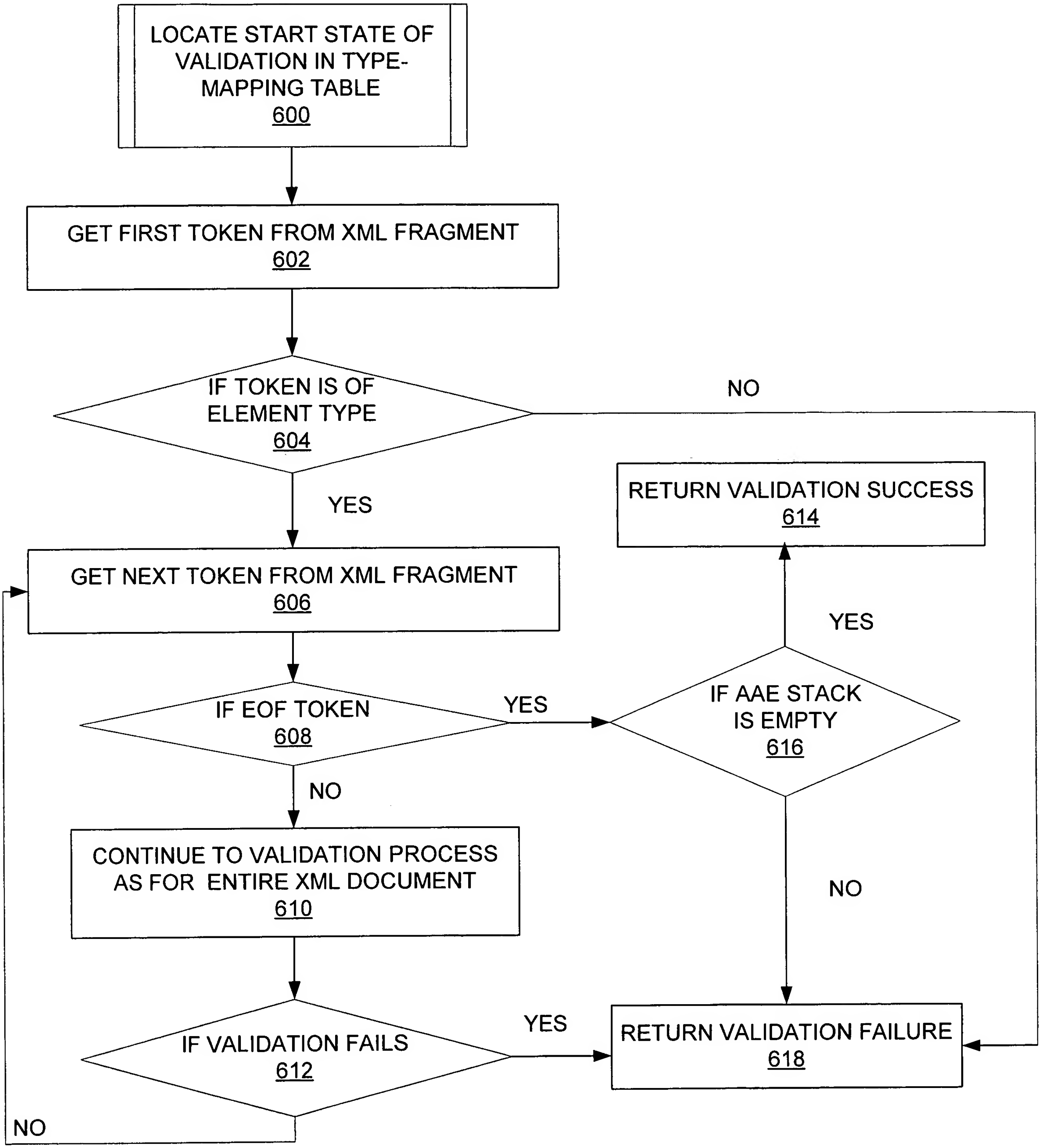

LOCATE START STATE OF VALIDATION IN TYPE-MAPPING TABLE
600
GET FIRST TOKEN FROM XML FRAGMENT
602
IF TOKEN IS OF ELEMENT TYPE
604
NO
YES
RETURN VALIDATION SUCCESS
614
GET NEXT TOKEN FROM XML FRAGMENT
606
YES
IF EOF TOKEN
608
YES
IF AAE STACK IS EMPTY
616
NO
CONTINUE TO VALIDATION PROCESS AS FOR ENTIRE XML DOCUMENT
610
NO
YES
IF VALIDATION FAILS
612
RETURN VALIDATION FAILURE
618
NO

FIGURE 6

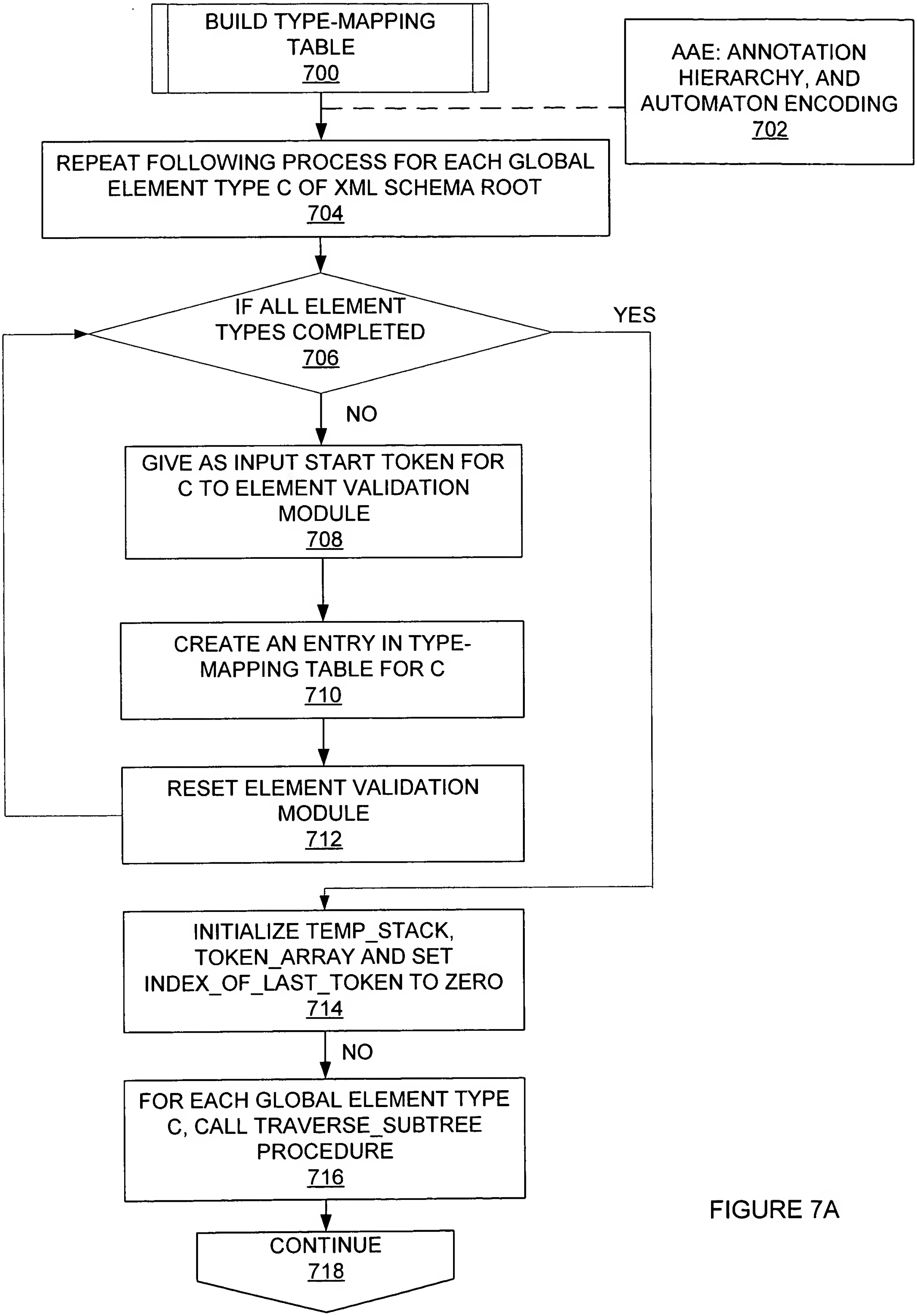

BUILD TYPE-MAPPING TABLE
700
AAE: ANNOTATION HIERARCHY, AND AUTOMATON ENCODING
702
REPEAT FOLLOWING PROCESS FOR EACH GLOBAL ELEMENT TYPE C OF XML SCHEMA ROOT
704
IF ALL ELEMENT TYPES COMPLETED
706
YES
NO
GIVE AS INPUT START TOKEN FOR C TO ELEMENT VALIDATION MODULE
708
CREATE AN ENTRY IN TYPE-MAPPING TABLE FOR C
710
RESET ELEMENT VALIDATION MODULE
712
INITIALIZE TEMP_STACK, TOKEN_ARRAY AND SET INDEX_OF_LAST_TOKEN TO ZERO
714
NO
FOR EACH GLOBAL ELEMENT TYPE C, CALL TRAVERSE_SUBTREE PROCEDURE
716
CONTINUE
718

FIGURE 7A

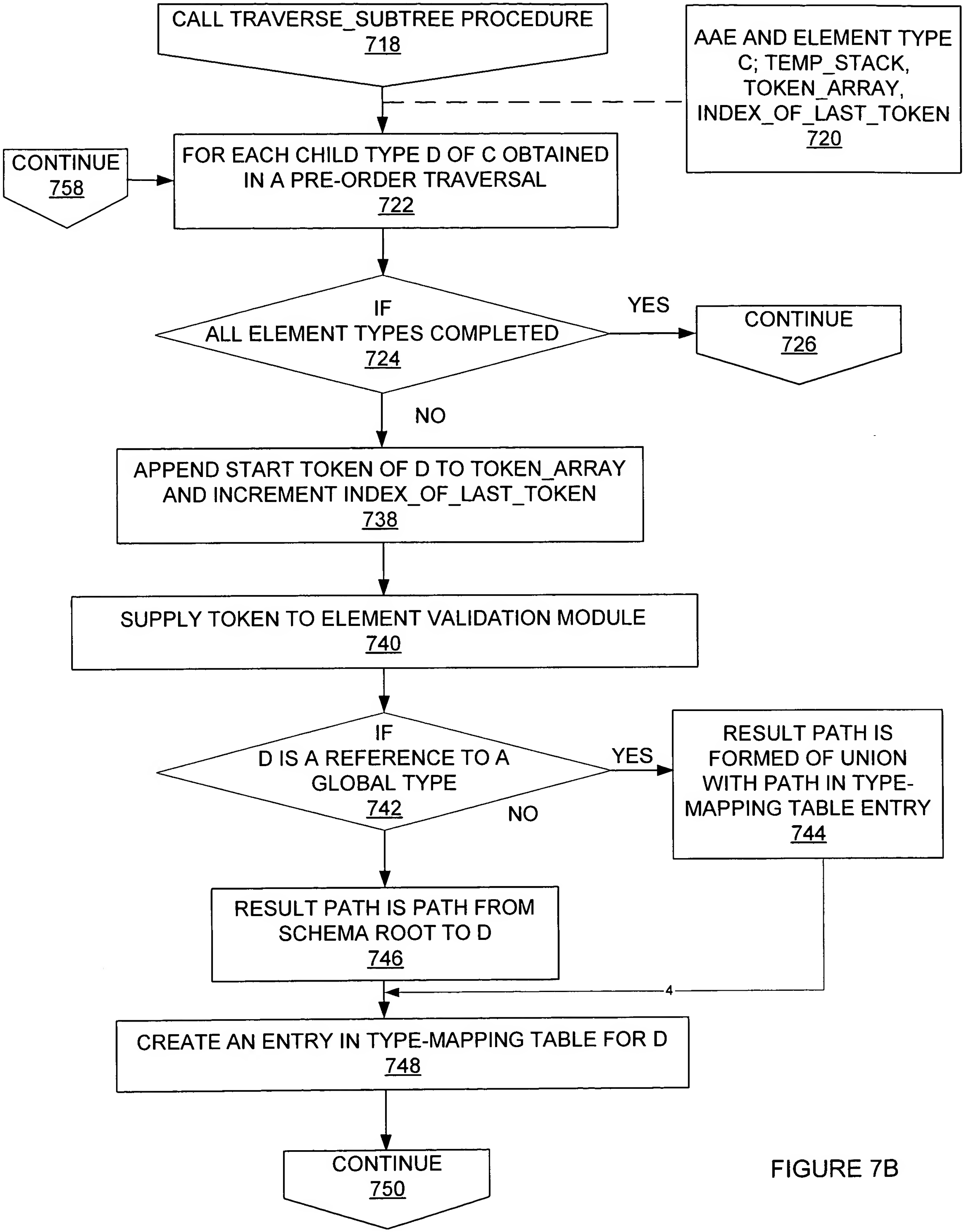

CALL TRAVERSE_SUBTREE PROCEDURE
718
AAE AND ELEMENT TYPE C; TEMP_STACK, TOKEN_ARRAY, INDEX_OF_LAST_TOKEN
720
CONTINUE
758
FOR EACH CHILD TYPE D OF C OBTAINED IN A PRE-ORDER TRAVERSAL
722
IF ALL ELEMENT TYPES COMPLETED
724
YES
CONTINUE
726
NO
APPEND START TOKEN OF D TO TOKEN_ARRAY AND INCREMENT INDEX_OF_LAST_TOKEN
738
SUPPLY TOKEN TO ELEMENT VALIDATION MODULE
740
IF D IS A REFERENCE TO A GLOBAL TYPE
742
YES
RESULT PATH IS FORMED OF UNION WITH PATH IN TYPE-MAPPING TABLE ENTRY
744
NO
RESULT PATH IS PATH FROM SCHEMA ROOT TO D
746
4
CREATE AN ENTRY IN TYPE-MAPPING TABLE FOR D
748
CONTINUE
750

FIGURE 7B

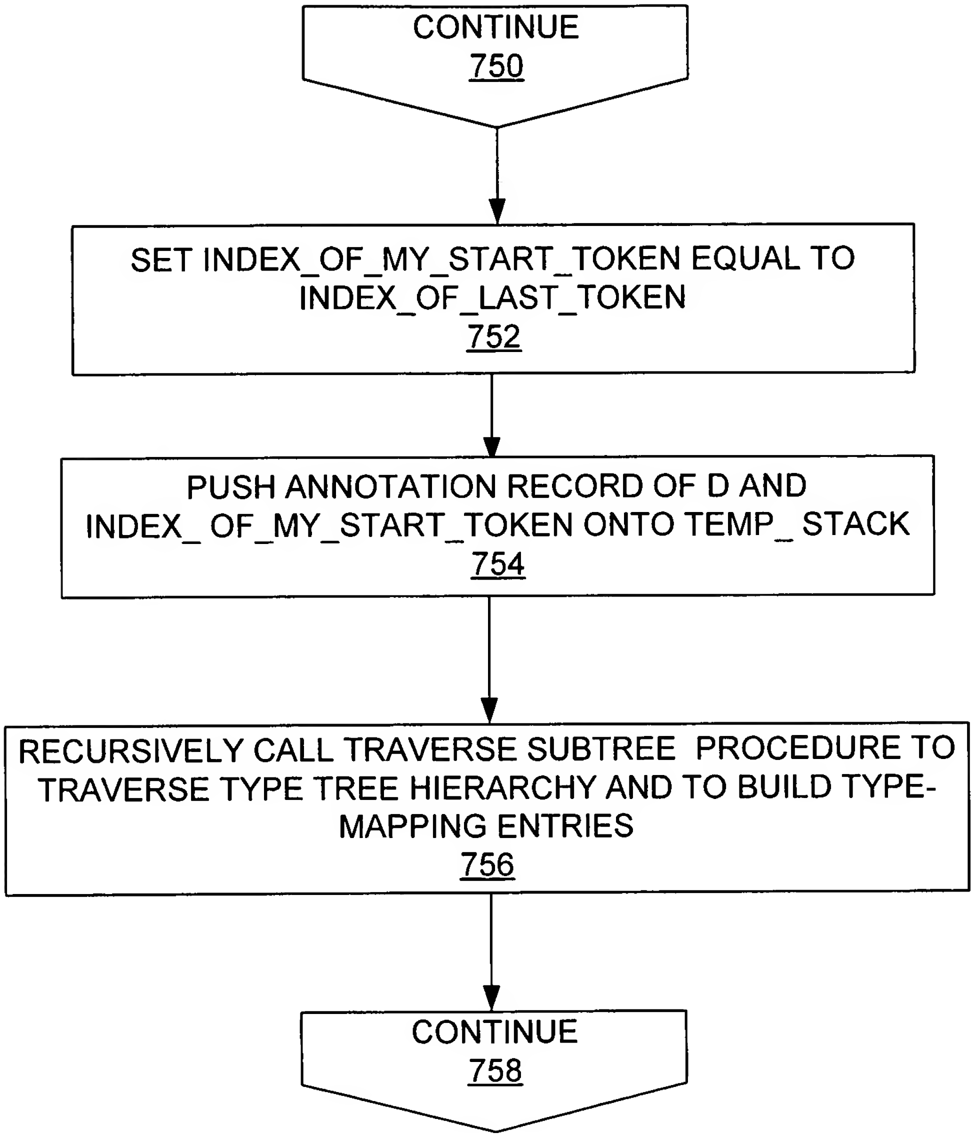

CONTINUE
750
SET INDEX_OF_MY_START_TOKEN EQUAL TO INDEX_OF_LAST_TOKEN
752
PUSH ANNOTATION RECORD OF D AND INDEX_ OF_MY_START_TOKEN ONTO TEMP_ STACK
754
RECURSIVELY CALL TRAVERSE SUBTREE PROCEDURE TO TRAVERSE TYPE TREE HIERARCHY AND TO BUILD TYPE-MAPPING ENTRIES
756
CONTINUE
758

FIGURE 7C

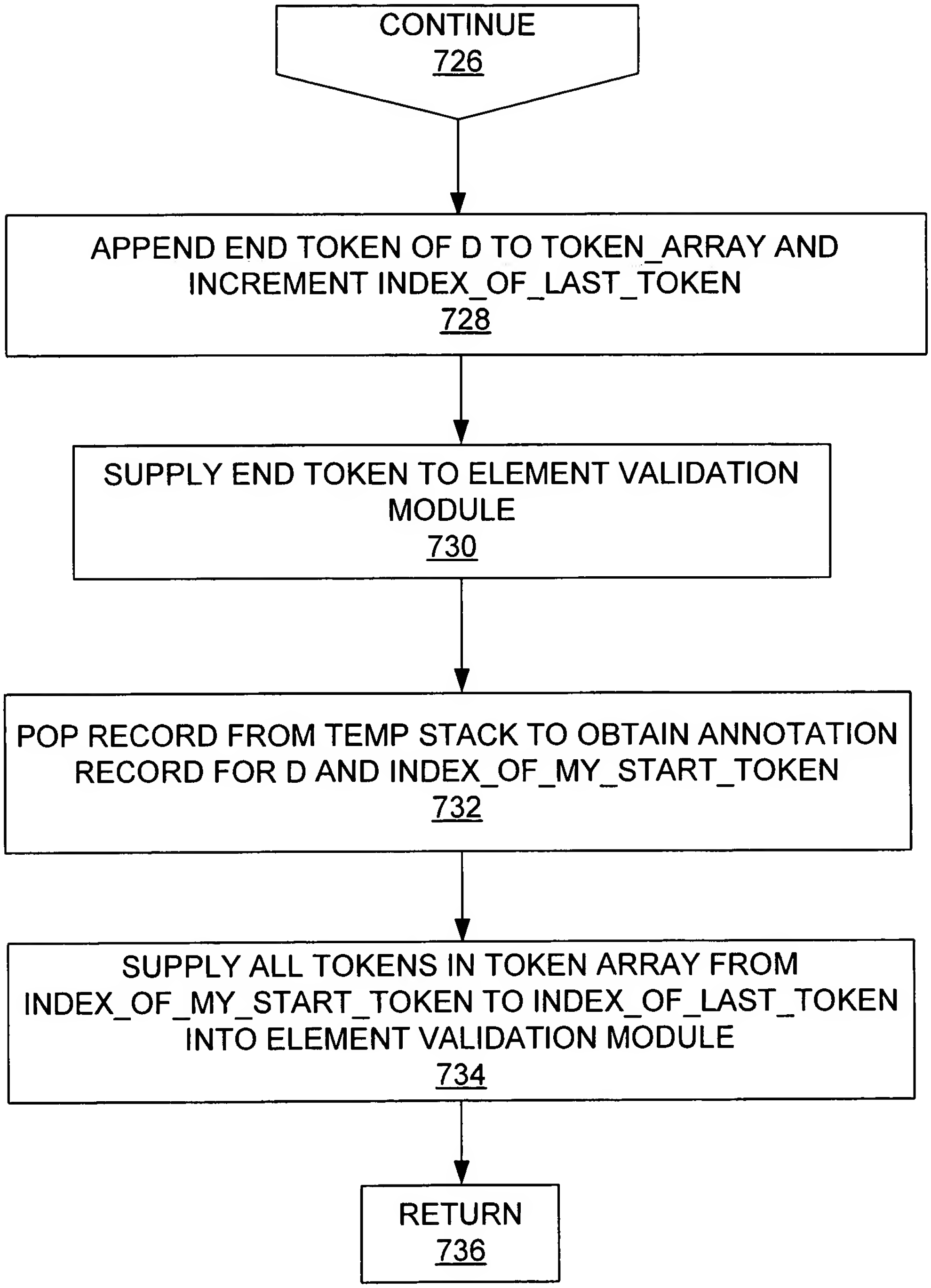
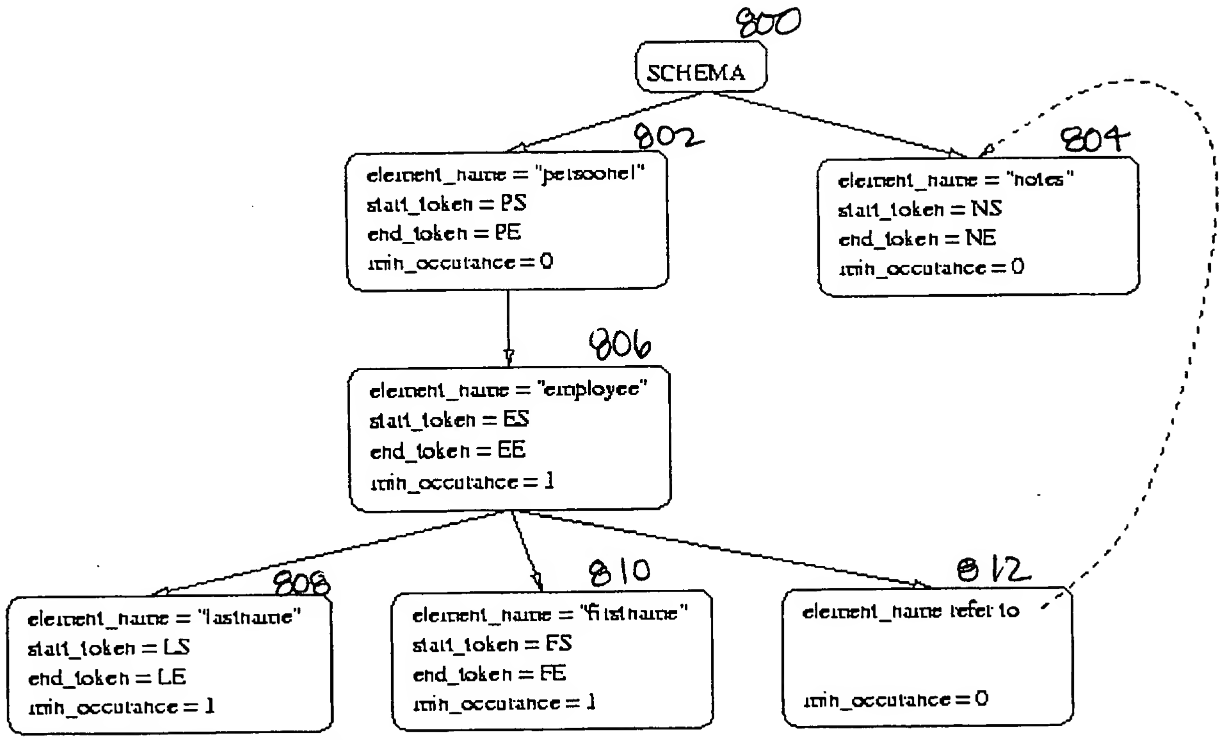

CONTINUE
726
APPEND END TOKEN OF D TO TOKEN_ARRAY AND INCREMENT INDEX_OF_LAST_TOKEN
728
SUPPLY END TOKEN TO ELEMENT VALIDATION MODULE
730
POP RECORD FROM TEMP STACK TO OBTAIN ANNOTATION RECORD FOR D AND INDEX_OF_MY_START_TOKEN
732
SUPPLY ALL TOKENS IN TOKEN ARRAY FROM INDEX_OF_MY_START_TOKEN TO INDEX_OF_LAST_TOKEN INTO ELEMENT VALIDATION MODULE
734
RETURN
736

FIGURE 7D



Type hierarchy ordered tree.

Figure 8

2. ~~The following algorithm will traverse the above type hierarchy ordered tree once and create the type mapping table.~~

```
TRAVERAL_TYPE_HIERARCHY_TREE (tree node: C,  Array of Token: Token_array, Int: index_of_last_token)
 {
    for (each child node D of C in schema order)
    {
        APPEND start_token of D to the Token_array.
        INCREASE index_of_last_token such that it points to the appended token in token_array.
        FEED the above token into the PushdownAutomaton.
        if (D is a reference to a global element type)
        {
            /* c */
            Union the path from SCHEMA to D with the path in the typemappying entry of the global element type.
            replace the original path with the result path above.
        }else
            create an entry <the path from SCHEMA to D, the element name of D, type annotation of D, the current state of the
PushdownAutomaton>.
        my_start_token_index := index_of_last_token.
        PUSH the annotation of D and my_start_token_index  onto Tmp_stack.
        TRAVERAL_TYPE_HIERARCHY_TREE (D, Token_array, index_of_last_token);
```

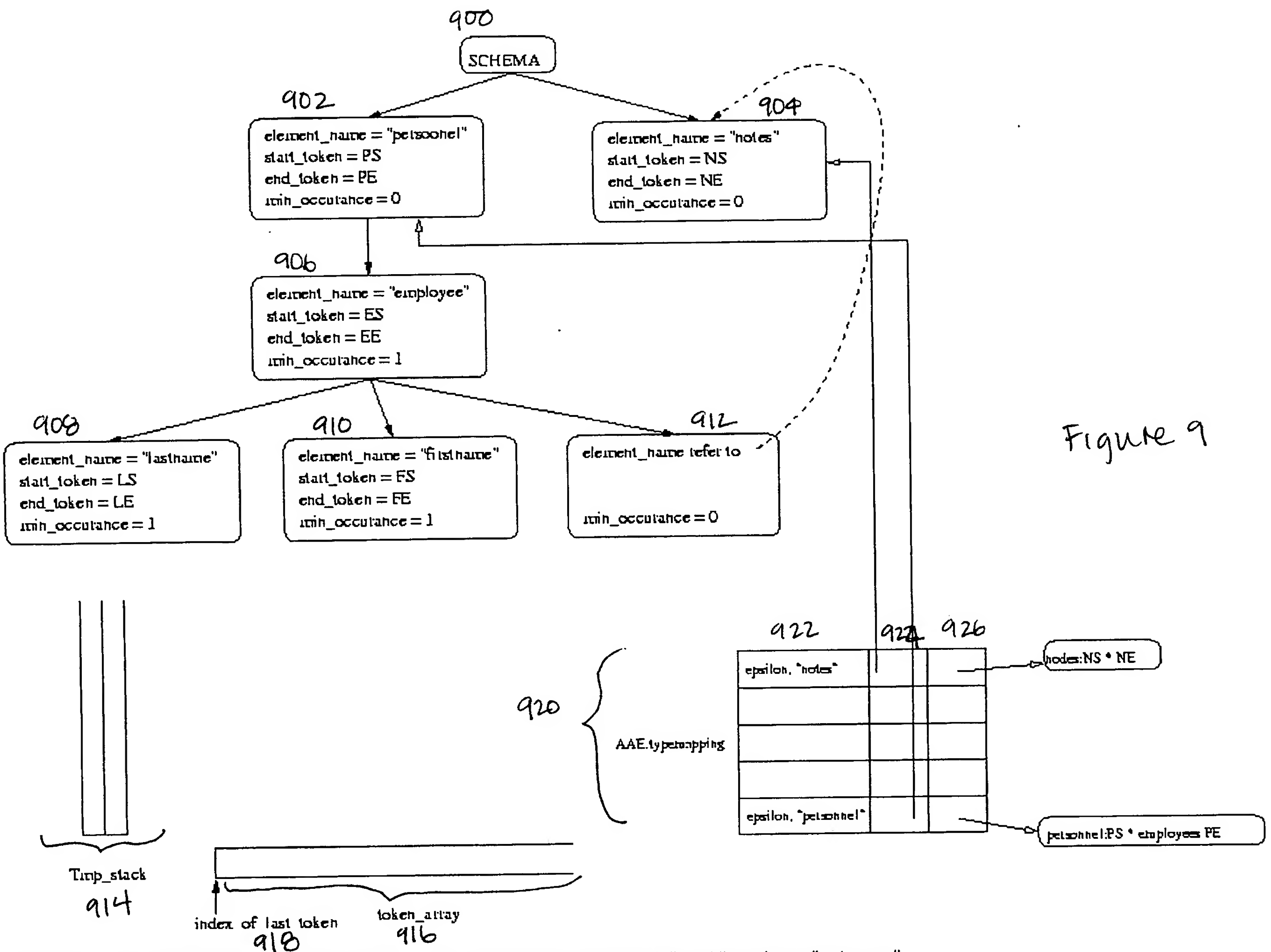
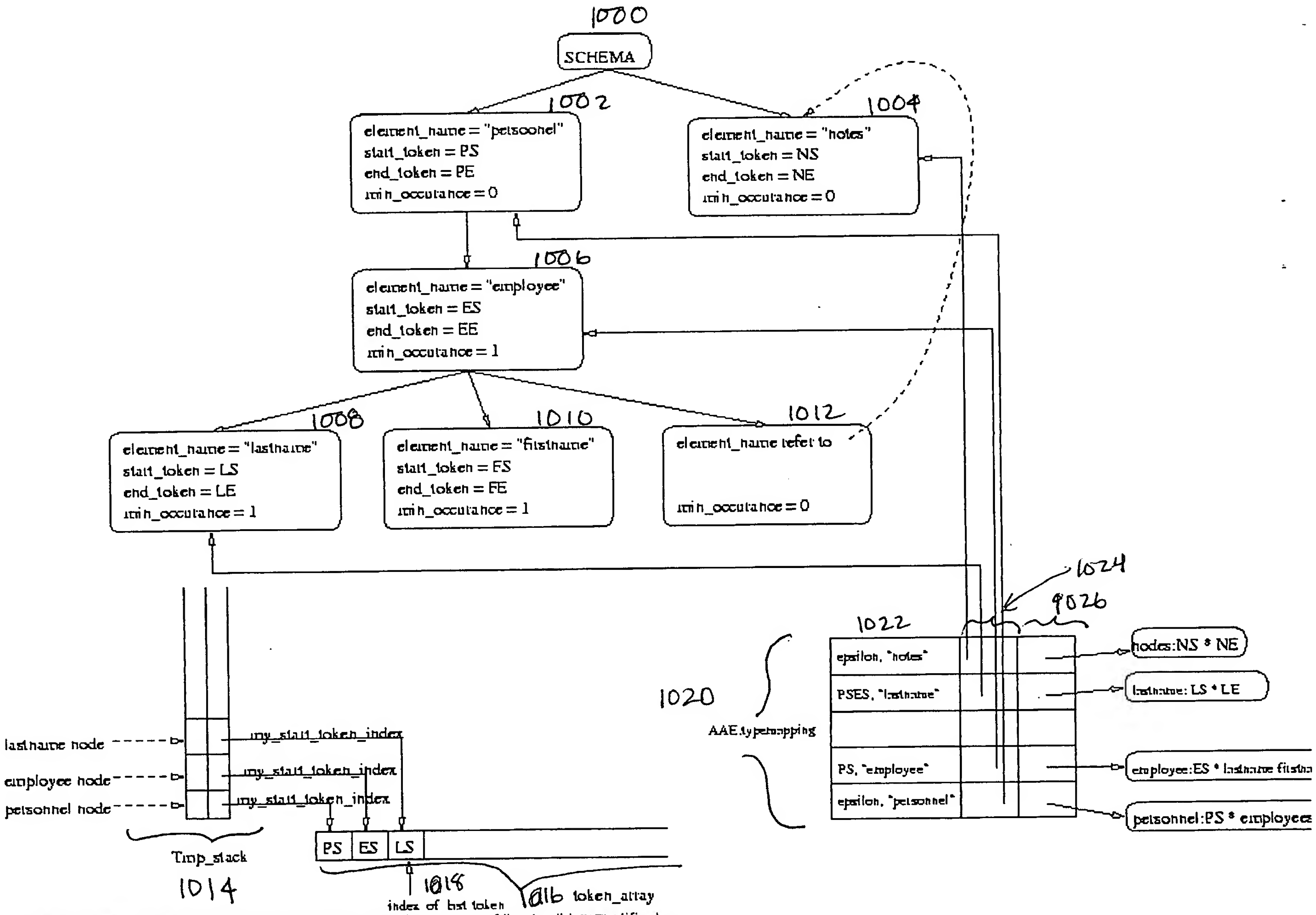


Figure 9

~~b. Second, by traversing the tree, we create the type mappings of "employee" and "employee/lastname"~~.



~~c. Lastly, we show how the element type mapping entry of "notes" is modified.~~

Figure 10

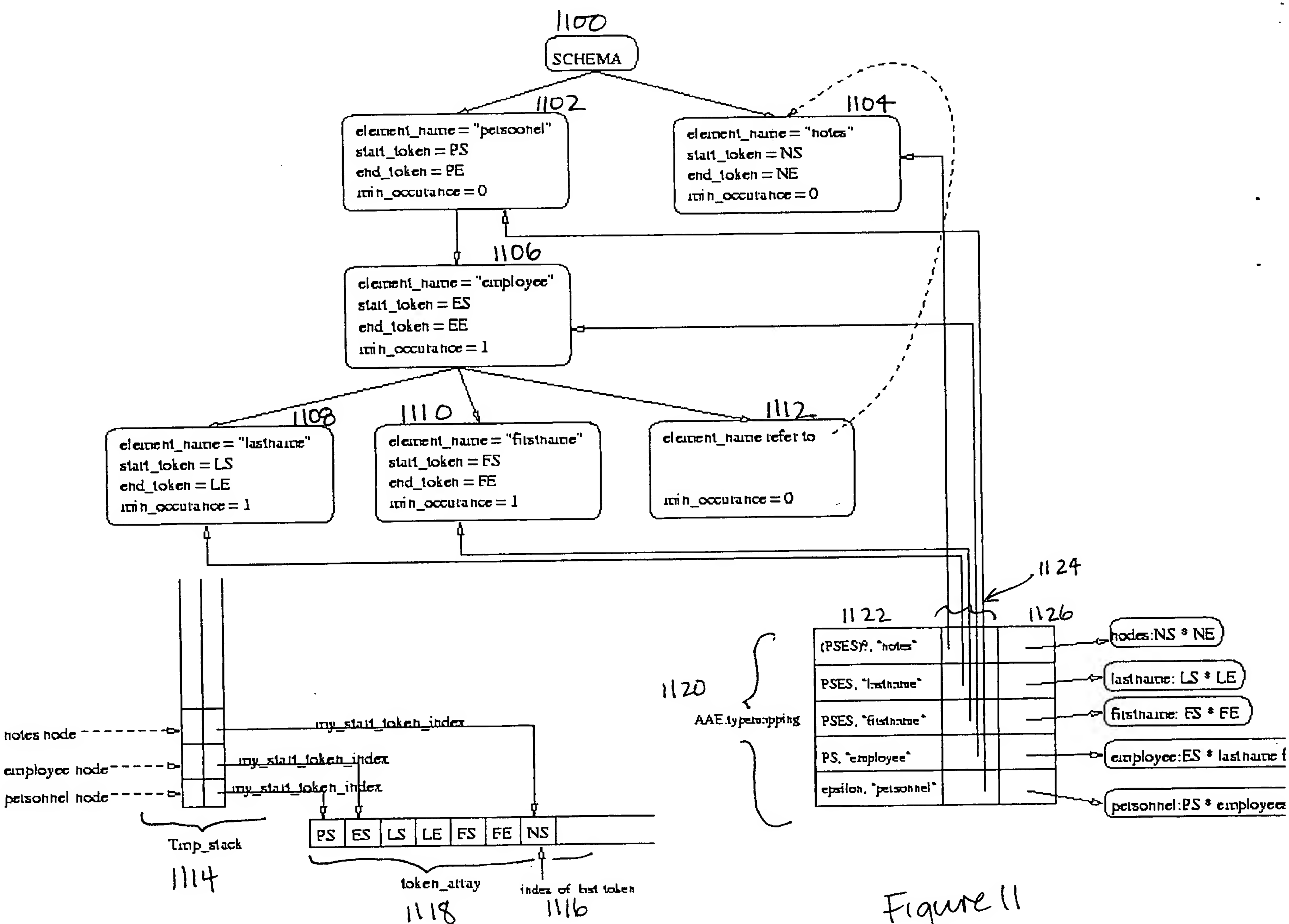

1100
SCHEMA
1102
element_name = "personnel"
start_token = PS
end_token = PE
min_occurance = 0
1104
element_name = "notes"
start_token = NS
end_token = NE
min_occurance = 0
1106
element_name = "employee"
start_token = ES
end_token = EE
min_occurance = 1
1108
element_name = "lastname"
start_token = LS
end_token = LE
min_occurance = 1
1110
element_name = "firstname"
start_token = FS
end_token = FE
min_occurance = 1
1112
element_name refer to
min_occurance = 0
notes node
employee node
personnel node
my_start_token_index
my_start_token_index
my_start_token_index
Tmp_stack
1114
PS
ES
LS
LE
FS
FE
NS
token_array
1118
index of last token
1116
1120
AAE.typemapping
1122
1124
1126
(PSES)?, "notes"
PSES, "lastname"
PSES, "firstname"
PS, "employee"
epsilon, "personnel"
notes:NS * NE
lastname: LS * LE
firstname: FS * FE
employee:ES * lastname f
personnel:PS * employee

Figure 11